


Blog

Asynchronous APIs in Choreographed Microservices (<https://nordicapis.com/asynchronous-apis-in-choreographed-microservices/>)

POSTED BY FRANCISCO MÉNDEZ VILAS ([HTTPS://NORDICAPIS.COM/AUTHOR/FMVILAS/](https://nordicapis.com/author/fmvilas/)) | OCTOBER 18, 2016DESIGN ([HTTPS://NORDICAPIS.COM/API-INSIGHTS/DESIGN/](https://nordicapis.com/api-insights/design/))

 Facebook (<https://www.facebook.com/sharer/sharer.php?u=https://nordicapis.com/asynchronous-apis-in-choreographed-microservices/&t=Asynchronous+APIs+in+Choreographed+Microservices>)


20  Twitter


3  Google+ (<https://plus.google.com/share?url=https://nordicapis.com/asynchronous-apis-in-choreographed-microservices/>)

136

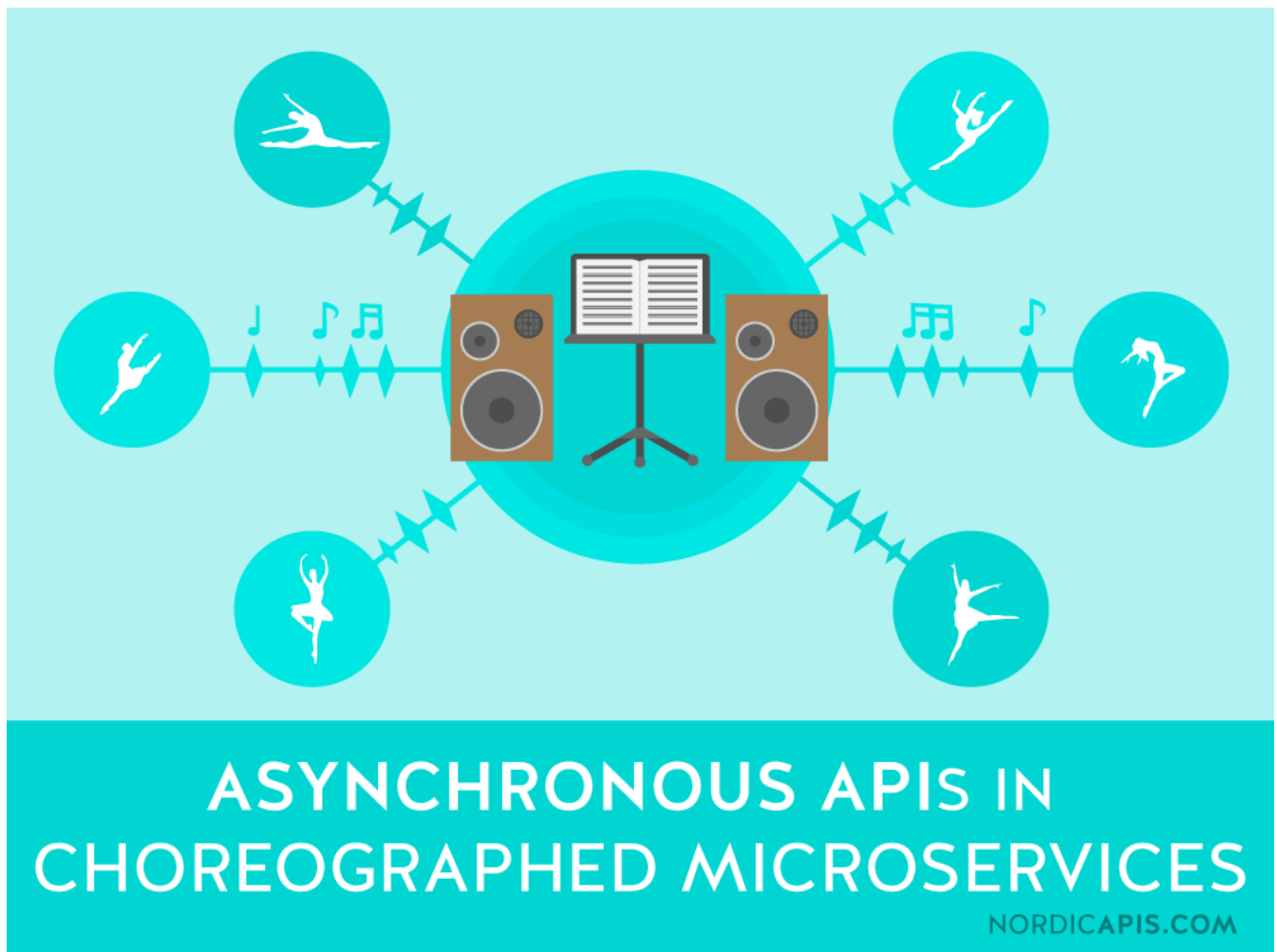
 LinkedIn ([https://www.linkedin.com/shareArticle?](https://www.linkedin.com/shareArticle?mini=true&ro=true&trk=EasySocialShareButtons&title=Asynchronous+APIs+in+Choreographed+Microservices&url=https://nordicapis.com/asynchronous-apis-in-choreographed-microservices/)

<https://www.linkedin.com/shareArticle?mini=true&ro=true&trk=EasySocialShareButtons&title=Asynchronous+APIs+in+Choreographed+Microservices&url=https://nordicapis.com/asynchronous-apis-in-choreographed-microservices/>)

 Reddit (<http://reddit.com/submit?url=https://nordicapis.com/asynchronous-apis-in-choreographed-microservices/&title=Asynchronous+APIs+in+Choreographed+Microservices>)

 HackerNews (<https://news.ycombinator.com/submitlink?u=https://nordicapis.com/asynchronous-apis-in-choreographed-microservices/&t=Asynchronous+APIs+in+Choreographed+Microservices>)

Total: 159



When building **microservices** you have to first make a decision: How am I going to manage service-to-service communication? Most developers will answer immediately: API calls. But then, more questions come to mind:

- Am I going to call services directly?
- Doesn't it block the caller service?
- Doesn't it introduce a dependency between both services?
- What if the called service is down?

- What if I have multiple instances of the called service and want to balance the workload?

Hopefully you'll ask yourself these questions before you implement your microservices architecture, as these are problems that need to be solved. But, have you ever realized that the root cause of all these problems is just one? We're trying to do asynchronous stuff on top of a synchronous protocol.

So let's ask ourselves again: How am I going to manage service-to-service communication? Let me answer this time: **Asynchronous API calls**.

Asynchronous APIs

Don't get confused with non-blocking API calls. In the Asynchronous APIs paradigm there's no client or server. Everything is **server-to-server**, or service-to-service if you prefer, and any service at any time could be the initiator of the communication.

You've already been doing it for some years: WebHooks.

WebHooks

Why were **WebHooks** invented? If you think about it, they are just another way of using HTTP APIs. They came to solve the basic problem of asynchronicity: "Another service might have something to tell me, but I don't know when, so I can't be asking forever."

The problem with WebHooks is they do direct calls, introducing a **dependency** between both services, and making it difficult for every service to manage failure in case the called service is down. WebHooks are essentially just a workaround on top of a widely used synchronous protocol. WebHooks are helpful, but they are overkill and largely unnecessary when it comes to internal service-to-service communication.

You then try to find something better. It can't block your service but it must let you have bi-directional communication out of the box. What about WebSockets?

WebSockets

Admit it. You felt free the first time you used **WebSockets**. They're really easy to use and you have a permanent bi-directional channel to communicate between systems. However nothing is perfect and you soon realized you have to define how the message exchange is going to be done. Message syntax and structure, channel names, etc.; all you had already defined by using REST APIs is lost. There's no standard methodology for guidance on this type of work. On top of that, whatever you decide, it must scale, be adaptable to future changes, and must be interoperable.

If you have used WebSockets in production then likely one of the first problems you faced was "too many open connections." Yes, you are self DDOS-attacking your system. You need a way to decide when a connection should be closed or it will live forever there.

WebSockets are a good option because they use an **HTTP Upgrade** request for the handshake, so most of the HTTP servers out there will understand. This increases interoperability.

However, it's how we use them that is concerning. The cool thing is that we're not blocking anymore and any service can initiate the communication, but, are you going to connect directly to the service? Isn't it introducing a dependency? What if the service is down? What if you want to distribute workload? You still have to fix these problems.

Letting the services talk to each other directly causes your infrastructure to be tightly coupled, so instead let's try to avoid that by implementing a choreographed microservices architecture.

A promotional banner for the Nordic APIs 2016 Platform Summit. The background is a teal color with faint, light-blue icons of various API-related concepts like databases, networks, and servers. On the left, there is a white square containing a blue bar chart icon and the text "NORDIC APIS". To the right of this, the text "Nordic APIs 2016 Platform Summit" is written in a large, white, sans-serif font. Below this, in a smaller white font, is "OCT 24-26 | GLOBEN | STOCKHOLM". Further down, the text "Join us for our largest conference to date!" is written in white. At the bottom center, there is a prominent orange button with the text "REGISTER NOW!" in white. On the right side of the banner, there is a stylized illustration of a construction site with a crane lifting a large block labeled "API" into place, with other construction elements like scaffolding and a truck visible.

(<http://nordicapis.com/events/2016-platform-summit/>)

For talks on Asynchronous APIs and more, attend the Platform Summit!

Choreographed Microservices

In a **choreographed microservices architecture**, every service is independent. Just like dancers performing a choreography, they don't need to talk to each other but instead they listen to the same thing: the music. They all agree on following the same rhythm, beats, and tempo. They know what to do and when to do it, even if they don't perform the same moves. The result is a perfect dance performance.

So what do you need to get your microservices *dance performance* up and running?

- **The services** – *the dancers*
- **The message broker** – *the music*
- **The messages** – *the notes*
- **The topics or channels** – *the sound system*

The most important piece in a choreographed microservices architecture is the **message broker**. The services don't talk to each other, but instead they just connect to the message broker and publish/subscribe to certain topics or channels.

With this architecture you solve a previous problem. Now, dependency is partially removed because services don't talk to each other directly. However we still have to completely solve another problem: distributing the workload across multiple instances of the same service.

The Message Broker

The simplest form of a message broker is a system that routes messages between services. Luckily nowadays we can find message brokers with much more functionalities: RabbitMQ (<https://www.rabbitmq.com/>), Apache Kafka (<http://kafka.apache.org/>), NATS (<https://nats.io/>), etc.

One of the most useful features are **message queues**. Whenever you connect to the message broker you're actually connecting to a message queue. So if a service sends a message to another and the last one is down, the message will get enqueued and will be available once the service is up again. That solves the problem of having to deal with service failures.

Another cool feature is **exchanges**. Exchanges act like small message brokers inside the message broker. So, for instance, an exchange can have many queues connected to it, and, depending on its type, it can enqueue the message into the less busy one. This enables us to solve our problem about distributing the workload across multiple instances of the same service.

Notice how we've avoided using the following words: "request", "response", "event", and "action." The message broker deals with just that; messages.

The Messages

Messages are just a piece of information and don't have an implied meaning. A message can contain anything in any format. They usually have headers and a body or payload, however it will change depending on the protocol you choose.

What you put in a message can have a huge impact on your architecture. Before we said that using a broker **partially** removes the dependency between services. It's true that the dependency has been removed at the infrastructure level, however it might not be true at a program logic level.

If you keep using the messages as a request/response imitation then your program logic will have a dependency because it's somehow waiting for the response. While this is fine in some cases, I invite you to think about the problems in a different way. Whenever possible, don't think about sending orders (like `GET` or `POST` in HTTP) but instead try to send **events**. **Tell the system what happened and not what it has to do**. For instance, after a user signs up to your application, send a message saying: "a user has just signed up" and information about them. By doing so, the Email service will be notified and will send a welcome email to the user. But also the Metrics service will be notified and will register the signup into Intercom, Google Analytics, or elsewhere.

Thinking about events instead of actions lets us effectively decouple our logic. However sometimes it will be impossible or too complicated to model your system simply using events. In that case it's recommended you use a different exchange with a different topic structure, so you'll instantly be able to differentiate between **actions** and **events**.

Always sign the messages with a **unique key** – especially with actions. This will help you debug your system and emit a message as the response of another one, using a different topic. How carefully you define your topics is as important as what you put in your messages.

The Topics

Comparable to URLs to some extent, **topics** are like channels or routes. When you send a message, you send it to a certain topic, and when you create an exchange you can route it to certain topics.

If you're building asynchronous APIs in choreographed microservices, it's strongly recommended to use **AMQP** and/or **MQTT** protocols. Aside from many cool features you'll get out of the box, their topics (<http://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices>) structure is really powerful.

Though, you should be using a **naming convention** to tidy things up. In the end they are like routes in an HTTP API – if you follow REST conventions they will be more readable and predictable. You can define the naming convention you prefer, but I would recommend it has at least 2 things:

- **Use versioning:** `v1.user.signup`. If the message structure changes you don't have to break anything. Messages with the new structure will use `v2` and the old ones will use `v1`. It will allow you to migrate the services without problems.
- **Use present tense for actions and past tense for events:** `v1.user.signup` when you want to sign the user up in your system. `v1.user.signedup` when the process has finished. That way, at a first glance, you will instantly know if it was an action or an event.

Conclusion


In the last years we've seen huge growth in the Microservices field, however most of the engineers didn't want to replace the HTTP protocol for service-to-service communication, even when it proved to be harder to maintain and to scale the whole architecture. The lack of knowledge, information, or tools makes starting with a fresh protocol daunting.

Hitch (<https://hitchhq.com>), for example, decided to bet on this architecture, and they're attempting to go fully asynchronous behind the scenes. It's proving to be easier to maintain and to develop new features.

Is your organization using this approach? Attend the upcoming Platform Summit (<http://nordicapis.com/events/2016-platform-summit/>) to see a talk on this subject by Francisco Méndez Vilas, lead engineer at Hitch hq. He'll be discussing how to get started with async APIs in choreographed microservices.

Further resources

- HermesJS – Real-time messaging framework (<https://github.com/hitchhq/hermes>)
- Runnable – Ponos (<https://github.com/Runnable/ponos>)
- Event-driven Microservices Using RabbitMQ (<http://blog.runnable.com/post/150022242931/event-driven-microservices-using-rabbitmq>)
- MQTT Essentials (<http://www.hivemq.com/blog/mqtt-essentials/>)
- RabbitMQ tutorials (<https://www.rabbitmq.com/getstarted.html>)

 Facebook (<https://www.facebook.com/sharer/sharer.php?u=https://nordicapis.com/asynchronous-apis-in-choreographed-microservices/&t=Asynchronous+APIs+in+Choreographed+Microservices>)


20  Twitter


3  Google+ (<https://plus.google.com/share?url=https://nordicapis.com/asynchronous-apis-in-choreographed-microservices/>)

136


 LinkedIn ([https://www.linkedin.com/shareArticle?](https://www.linkedin.com/shareArticle?mini=true&ro=true&trk=EasySocialShareButtons&title=Asynchronous+APIs+in+Choreographed+Microservices&url=https://nordicapis.com/asynchronous-apis-in-choreographed-microservices/)

<https://www.linkedin.com/shareArticle?mini=true&ro=true&trk=EasySocialShareButtons&title=Asynchronous+APIs+in+Choreographed+Microservices&url=https://nordicapis.com/asynchronous-apis-in-choreographed-microservices/>)

 Reddit (<http://reddit.com/submit?url=https://nordicapis.com/asynchronous-apis-in-choreographed-microservices/&title=Asynchronous+APIs+in+Choreographed+Microservices>)

 HackerNews (<https://news.ycombinator.com/submitlink?u=https://nordicapis.com/asynchronous-apis-in-choreographed-microservices/&t=Asynchronous+APIs+in+Choreographed+Microservices>)

Total: 159

 api (<https://nordicapis.com/tag/api/>), APIs (<https://nordicapis.com/tag/apis/>), architecture (<https://nordicapis.com/tag/architecture/>), asynchronous (<https://nordicapis.com/tag/asynchronous/>), Asynchronous API (<https://nordicapis.com/tag/asynchronous-api/>), beat (<https://nordicapis.com/tag/beat/>), choreographed (<https://nordicapis.com/tag/choreographed/>), choreographed microservices architecture (<https://nordicapis.com/tag/choreographed-microservices-architecture/>), choreography (<https://nordicapis.com/tag/choreography/>), client (<https://nordicapis.com/tag/client/>), dance (<https://nordicapis.com/tag/dance/>), Hitch (<https://nordicapis.com/tag/hitch/>), HTTP (<https://nordicapis.com/tag/http/>), Messages (<https://nordicapis.com/tag/messages/>), microservice (<https://nordicapis.com/tag/microservice/>), microservices (<https://nordicapis.com/tag/microservices/>), music (<https://nordicapis.com/tag/music/>), performance (<https://nordicapis.com/tag/performance/>), Platform Summit (<https://nordicapis.com/tag/platform-summit/>), server (<https://nordicapis.com/tag/server/>), server-to-server (<https://nordicapis.com/tag/server-to-server/>), service-to-service (<https://nordicapis.com/tag/service-to-service/>), tempo (<https://nordicapis.com/tag/tempo/>), Topics (<https://nordicapis.com/tag/topics/>), WebHooks (<https://nordicapis.com/tag/webhooks/>), WebSockets (<https://nordicapis.com/tag/websockets/>)

2 Comments


(https://nordicapis.com/asynchronous-apis-in-choreographed-microservices/#disqus_thread)



About Francisco Méndez Vilas

Fran is the Lead Engineer at Hitch. During the last year he's been investigating about the present and future of APIs and how they will drive the Microservices and IoT revolutions. Previously, he worked for Redbooth as a full-stack engineer. When he's not at work you can find him drinking cold beer or lost in any place of the world (or both).

 (<https://nordicapis.com/author/fmvilas/>)  (<https://twitter.com/fmvilas>)

 (<https://www.linkedin.com/in/fmvilas>)

 Virtualization, Sandboxes,... (<https://nordicapis.com/virtualization-sandboxes-playgrounds-wholesome-api/>)

API Longevity, Devious...  (<https://nordicapis.com/api-longevity-devious-drones-bourbon-insights-2016-platform-summit/>)

2 Comments Nordic APIs

Login

Recommend 4 Share

Sort by Best



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name



bekasik • 2 years ago

We have numerous ways to version (in url, etc.) and document (swagger, etc.) REST and other synchronous APIs. What about versioning and documenting messaging APIs? Had no luck discovering any standards/implementations.

^ | v • Reply • Share ›



fmvilas • bekasik • a year ago

Hi bekasik,

Sorry for the late response. I'm right now working on a OpenAPI-like standard for Asynchronous APIs. I'll post more info about it soon.

In the meantime you might want to join the Async APIs channel on Slack: <https://async-apis-slack.he...>

Thanks!

1 ^ | v • Reply • Share ›

ALSO ON NORDIC APIS

Tooling Review: AsyncAPI

1 comment • 6 months ago

fmvilas — Thanks for the great article Kristopher :)
Avatar

High-Grade API Security For Banks

2 comments • 5 months ago

Martin Flower — I wonder if "\$250 billion annually on cybersecurity" should
Avatar actually be \$500 million annually ?

Using JSON-LD To Establish Semantic Linked Data

3 comments • 9 months ago

Bill C. Doerrfeld — Thanks for sharing Scott! I'm inserting the link to Hydra
Avatar once more because the one above isn't going through - <http://www.hydra-cg.com/>

11+ Killer Open Data Sources and Free Visualization Tools

1 comment • a year ago

India — I would also suggest Enigma Public (<https://public.enigma.com/>)
Avatar as a great source for publicly available data.

[Subscribe](#) [Add Disqus to your site](#) [Add Disqus](#) [Add](#) [Disqus' Privacy Policy](#) [Privacy Policy](#) [Privacy](#)


(https://nordicapis.com/events/the-2018-platform-summit/)

NOMINATE THE BEST PUBLIC API 2018



(<https://nordicapis.com/best-public-api-of-2018/>)

CALL FOR **SPEAKERS** 2018

SUBMIT PROPOSAL NOW!

(<https://nordicapis.com/call-speakers-2018/>)

SMARTER TECH DECISIONS USING APIS

Subscribe to our mailing list

Subscribe

POPULAR POSTS



(<https://nordicapis.com/best-practices-api-error-handling/>) Best Practices for API Error Handling (<https://nordicapis.com/best-practices-api-error-handling/>)

by Kristopher Sandoval (<https://nordicapis.com/author/sandovaleffect/>) | posted on June 15, 2017



(<https://nordicapis.com/fintech-and-apis-making-a-bank-programmable/>) FinTech and APIs: Making the Bank Programmable (<https://nordicapis.com/fintech-and-apis-making-a-bank-programmable/>)

by Bill Doerrfeld (<https://nordicapis.com/author/billdoerrfeld/>) | posted on September 15, 2015



(<https://nordicapis.com/introduction-to-api-versioning-best-practices/>) Introduction to API Versioning Best Practices (<https://nordicapis.com/introduction-to-api-versioning-best-practices/>)

by Joshua Curry (<https://nordicapis.com/author/lucidbeaming/>) | posted on November 3, 2017



(<https://nordicapis.com/token-design-better-api-architecture/>) Token Design for a Better API Architecture (<https://nordicapis.com/token-design-better-api-architecture/>)

by Giovanni Casinelli (<https://nordicapis.com/author/giovanni-casinelli/>) | posted on January 2, 2016



(<https://nordicapis.com/austin-api-summit-wrap-up/>) Austin API Summit Wrap Up (<https://nordicapis.com/austin-api-summit-wrap-up/>)

by Bill Doerrfeld (<https://nordicapis.com/author/billdoerrfeld/>) | posted on June 15, 2018

RECENT POSTS

How To Design Frictionless APIs (<https://nordicapis.com/how-to-design-frictionless-apis/>)

8 Frameworks to Build A Web API In Scala (<https://nordicapis.com/8-frameworks-to-build-a-web-api-in-scala/>)

The Role of APIs In Blockchain (<https://nordicapis.com/the-role-of-apis-in-blockchain/>)

How Bad Developer Portals Kill APIs (<https://nordicapis.com/how-bad-developer-portals-kill-apis/>)

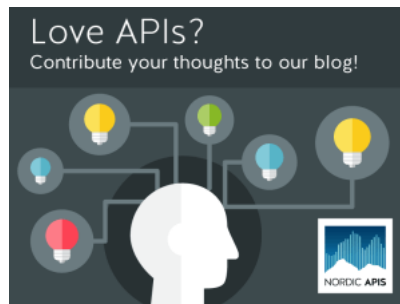
Austin API Summit Wrap Up (<https://nordicapis.com/austin-api-summit-wrap-up/>)

SUBSCRIBE TO OUR FEED



Nordic APIs RSS (<http://nordicapis.com/feed/>)

CREATE WITH US



(https://docs.google.com/a/twobotechnologies.com/forms/d/12Ng9A_QKUjmAHDgv8Pxb4uLlKECGJawV3vwAWJ4WxTs/viewform)



TWITTER (<HTTPS://TWITTER.COM/NORDICAPIS>)



FACEBOOK (<HTTPS://WWW.FACEBOOK.COM/NORDICAPIS>)



YOUTUBE (<HTTPS://WWW.YOUTUBE.COM/USER/NORDICAPIS>)



SLIDESHARE (<HTTP://WWW.SLIDESHARE.NET/NORDICAPIS>)



INSTAGRAM (<HTTPS://WWW.INSTAGRAM.COM/NORDICAPIS/>)



RSS (<HTTP://NORDICAPIS.COM/FEED/>)